

A Young Player's Guide to Assignment 3

July 24, 2014

STOP!!! Please ensure that you've read the formal Assignment 3 requirements document before reading this guide.

We realise that this project is quite possibly the most ambitious piece of software development you have yet encountered. The purpose of this document is principally to set down some “traps for young players” in the form of a checklist. The intent is that you use the checklist to increase the polish and professionalism of your project (and perhaps more importantly, the marks you receive). There are also subsequent sections that expand on these points and provide advice on how to go about implementing them.

Project Checklist

Initial Proposal/Initial Design Document

- ☐ Should I quickly run my idea past Peter before writing up?
- ☐ Is my initial proposal a PDF document? (See Section 3.4)
- ☐ Does my document filename contain my name and student number? (See Section 3.4)

Design Document (See Section 4.3.1)

- ☐ Have I converted my proposal into the final design document sufficiently?
Have I..
 - ☐ included a real screenshot?
 - ☐ updated the functionality to represent the current scope?
 - ☐ updated my class design?

Reflection (See Section 4.3.4)

- ☐ Did I list all known bugs?
- ☐ Did I give consideration to alternatives?

Installation Notes (See Section 4.3.3)

- ☐ Do my installation notes have ...
 - ☐ the operating system I developed under?
 - ☐ the packages necessary to run the program? Do they require any special installation?
- ☐ Have I specified whether my program must be run through IDLE? OR
- ☐ Does my program run on the command line? If so,
 - ☐ Does my program need to be run from a particular directory (folder)?
 - ☐ Does my program require particular arguments or flags?
- ☐ Have I attempted to make my code cross platform?
 - ☐ Does my program only have relative paths?
 - ☐ Are **all** my filenames lowercase without spaces or special characters?
 - ☐ Do my filenames use the same capitalisation for files as well as within the code?
 - ☐ Have I used the os library to handle path slashes and slashes?
 - ☐ Have I enabled universal newline support for reading text files?

Demonstration

- ☐ Do I have a plan of how to sell my program and logically demonstrate its features? How will I conclude? (See Section 4.4.1)
- ☐ Have I tested the program on the computer I will be giving the demo on? (See Section 4.4.2)

Code

- ☐ Are any of the lines in my program longer than eighty characters (See Section 4.2.2)
- ☐ Do I have a top-level comment in each and every file? (See Section 4.2.2)
 - ☐ with my name?

- ☐ with my program name?
- ☐ with my program's current version?
- ☐ with what my program does?
- ☐ with license and copyright information?
- ☐ Do I have magic numbers in my code? (See Section 4.2.2)
- ☐ Do I have repeated code in my program? (See Section 4.2.2)
- ☐ Do I have any spaghetti code? (See Section 4.2.2)

Miscellaneous - User Interfaces (See Section 4.2.1)

- ☐ Does my program window have a title?
- ☐ Are my menus consistent with other programs? (eg. File, Edit, View, Help, etc)
- ☐ Is my help and about content well spaced and indented? Is the window too long?
- ☐ Are my buttons consistent with other programs? (eg. OK and Cancel - capitalisation, size, alignment and spacing are all standardised)
- ☐ Do my buttons do what they say? (eg. Will Cancel in a Save Dialog actually cancel?)
- ☐ Are the colours I've used what the user expects? (eg. a bright pink Cancel box could be jarring for a user)
- ☐ How well does my interface align with user expectations?
- ☐ Does my interface mainly require the user to use the keyboard or mouse? Or do they have to keep switching between them?
- ☐ Could there be aspects of my program that are counter-intuitive to people other than me?
- ☐ Could there be aspects of my program that may annoy the user? (eg. think popup ads)
- ☐ Could there be aspects of my program where the user feels lost? (eg. long delays with no feedback while something loads)

Contents

1	Staff Assistance	5
2	Assignment Expectations	5
2.1	Questions and Misconceptions	5
3	The Proposal	7
3.1	Approach	7
3.2	Be detailed	7
3.3	Decompose the project	7
3.4	Format	8
3.5	Project Ideas	8
3.5.1	Game Projects	8
3.5.2	Network/Web-based Projects	9
3.5.3	Image/Video Processing Projects	9
3.5.4	Multimedia Application	9
4	The Project	10
4.1	Project Management	10
4.1.1	Time Management	10
4.1.2	Backup and Version Control	10
4.2	Implementation	11
4.2.1	User Interfaces	11
4.2.2	Code Style	11
4.3	Documentation	13
4.3.1	Good Proposals help make good final documents	13
4.3.2	Start Early	13
4.3.3	Installation	13
4.3.4	Reflection	14
4.4	Demonstration	14
4.4.1	Plan your demo	15
4.4.2	Test under demo conditions	15

1 Staff Assistance

I urge you to talk to me (Peter) about your project - your ideas, problems you are having or if you just want to get some informal feedback over the course of your project.

Peter
Office: 78-316
Phone: (07) 3365-3461
Email: pjr@itee.uq.edu.au

2 Assignment Expectations

As mentioned previously, we realise that this project is quite possibly the most ambitious piece of software development you have yet encountered. Although this is a first year course, the type of marking scheme we use is quite different from that in other first year courses. We will assess you hard for you to find out where your limits are so you can grow as a Software Engineer - this doesn't mean we're trying to push you to past your breaking point or "that we're just out to fail you". See the proceeding section for some frequently asked questions and common misconceptions.

2.1 Questions and Misconceptions

I need to do this assignment in order to do better than pass!

No - you have to do it in order to get a 7 and it gives you some flexibility when aiming for a 6 but otherwise it is not necessary. In fact, if your aim is simply to pass then I advise against doing assignment 3 as it will distract you from doing a good job on assignment 2.

What's all this project ranking stuff about?

As stated in the assignment sheet, because each student chooses their own project, it is difficult to be prescriptive about the marking scheme. After running this assignment for a few semesters, we believe that a project grading is the best way to give you a feel for how the project might go. We want you to feel free to talk to us anytime about your project to keep it on track with your goals.

This assignment is too open-ended. There's no clear criteria for what I have to do.

Assignment 3 can be particularly stressful because of its complexity and its openness. We choose to leave Assignment 3 open to give students an opportunity to do something they are really interested in or something they really want to do. Some students can find the freedom that Assignment 3 provides a burden because the assignments vary from person to person and hence it is more difficult for you to get a sense of your progress than in the previous assignments which had generic criteria.

I've seen some of the examples on the course website. I could never do something that complex.

These are examples of the "best of the best" - the assignments received marks in the high 30s out of 40 marks. The students that received high marks were wholly passionate about their project - it was more than just an assignment but was something that they continued doing in their leisure time. It isn't necessary to go to these lengths for Assignment 3 in order to do well in the course.

Having said that, these projects may not have necessarily been "Five" projects when we approved their initial proposals however they showed us something with the "wow" factor on the day by adding more polish or more features. Don't be afraid to try, you may surprise yourself.

Isn't this course about programming? Why do I have to do all the other stuff?

Certainly one of the key outcomes of the course is learning the principles of object-oriented programming but remember that the course is not called Introduction to Programming but rather Introduction to Software Engineering which covers a greater area than programming. In the course of the assignment we expect you will spend many hours looking at documentation, planning your approach, diagnosing bugs and errors, strategising where on your project to focus your attention, considering how to design your user interface and others. All of these stand you in good stead for future software development. Unfortunately this can be obscured when you are in the thick of a difficult assignment - this is an appreciation that develops over time.

I've been putting all my time and energy into this assignment. My other courses are suffering.

As Assignment 3 students from previous semesters will tell you, time management is an extremely important aspect of this assignment. You need to make tradeoffs - it isn't necessary to get a perfect Assignment 3 in order to do well

in this course. We aren't looking at you to pour every spare minute into the assignment to the detriment of all your other courses.

I only received 19 out of 40 marks. I've never failed an assignment before.

19 out of 40 is not failure for Assignment 3. Understand that it is not necessary to produce a flawless Assignment 3 in order to get a 6 or a 7 overall - many students in previous semesters received 7s while still receiving around 20/40. We can appreciate that for a student looking for top marks, receiving 50% for a very difficult assignment can be very confronting but this does not constitute failure.

3 The Proposal

3.1 Approach

Many students will spend a few hours trying out an idea to see how workable it is for them. This is a reasonable approach. We do warn you however not to overcommit to your initial idea. Writing the entire assignment beforehand and reverse engineering the proposal from it is unwise as the project may require significant revision to be appropriate. Please feel free to sound us out on project ideas informally before submitting your initial proposal - it could save a lot of headaches later on. Also note that while we expect your final product will change from your submitted proposal, you must follow through with what you proposed or otherwise submit another proposal before the deadline. The project should not change so significantly as to seem a different project (eg. substituting a game for the web application proposed originally).

3.2 Be detailed

The more detailed you are about your project, what you are trying to achieve and how you intend to go about it, the more advice we can give you about its scope and methodology for implementing it.

3.3 Decompose the project

Even with the most enthusiasm and best of intentions, life can often get in the way of a serious project. It's a really good idea to break the project down into smaller components. Categorise these components into those that you will implement versus those that you may implement given enough time. You may even want to go as far as having a wishlist category for those really amazing

ideas that you think may be unrealistic to implement in the timeframe. This is useful because it helps you clarify your direction and give you priorities to focus on.

3.4 Format

We want your document in PDF. More specifically we will not accept .doc or docx formats. ITEE has support for printing PDFs (see <http://studenthelp.itee.uq.edu.au/printing/pdfprint.html>) There are also many free PDF virtual printers available for Windows, just search for PDF printer on download.com. Linux allows fairly straightforward printing of PDFs out of the box.

We would also appreciate if you could put your name and student number in the filename - dozens of files entitled `assign3proposal.pdf` can become confusing.

3.5 Project Ideas

You may have a firm idea of what you want to do for Assignment 3 in your mind. However if you're having difficulty coming up with a project idea you can take inspiration from programs that you use everyday (like writing a simple version of Word, Firefox, Photoshop or iTunes) or if you have a casual job you may like to write a tool to help you or your boss or you could ask yourself the question "What program could I write that would make my life easier?"

To help in choosing a project and to get a better idea of project rankings we now present some common choices for projects and what we expect from such projects. If you have some other idea for a project in mind then that's fine but it might be wise to run the idea past us before making a final decision.

For game projects pygame is typically used. For other projects that require a GUI you could use tkinter but you should also consider using more sophisticated GUI libraries like PyQt or wxPython. Such libraries are typically faster, look better and have a much more extensive collection of widgets than tkinter. The downside of using these libraries is that you will need to learn how to use them. On the other hand, the fact that you have had to understand how to use a new system is likely to increase the ranking of your project.

3.5.1 Game Projects

An arcade, puzzle or side-scrolling adventure or shoot-em-up game is a good candidate for a project. Such projects are typically a rank 3 project but can be extended in various ways that might improve the ranking. Some games are too simple, such as Golf, while other games, like Chess are either too simple (if it's just about managing moves) or too hard (if you are attempting to write your own AI).

Most student invent their own games. Some produce reasonably faithful reproductions of existing games such as Frogger. Others take existing games, like PacMan, and vary it in interesting ways such as providing additional behaviours for the ghosts or changing the ‘board’ on which the games is played.

Games typically include sound effects and also a sound track.

Depending on the kind of game there are various ways the ranking could be improved such as those listed below.

- Make it a multi-player game over the network
- Program a range of difficulty levels that users can choose before starting the game
- Increase the range or sophistication of AI (behaviours) for the ‘enemies’ (related to previous point)
- Add a ‘level editor’ so your friends (who aren’t Python programmers can add to or modify your game in various ways).

3.5.2 Network/Web-based Projects

An instant messaging or IRC client, web server, or multi-device file synchronization are good candidates for projects. The typical ranking for such projects is a 3 but a lot depends on what sort of ‘advanced features’ you provide.

Python has lots of interesting libraries that might help from basic sockets to APIs for Django, Google App Engine, MQTT and Twitter. An application that takes advantage of Google Maps might be worth considering.

3.5.3 Image/Video Processing Projects

An image processing application can make a good project. For example, a simplified version of PhotoShop or Gimp is an option. For this to be at least a rank 3 project it would require implementing some interesting filters or generators. Examples include edge detection, Gaussian blur and plasma generators.

One way of increasing the ranking would be to have a plug-in feature where users could write their own filters and generators (in Python) as is done in Gimp to extend your tool.

Using SimpleCV or OpenCV to do video processing can make an interesting project and that can be quite a high ranking project (depending on the application). An example of using video processing from a USB webcam is to detect certain kinds of object and so be used as a sensor for a robot. One student involved in a research project on Zebra fish used post-processing of videos of the fish swimming to analyse movement and tail angles of the fish.

3.5.4 Multimedia Application

There are a wide range of reasonable projects in this area. A system for managing and playing your audio/video files is one example that would probably be a rank 3 project and has potential for a higher ranking project if enough features are added to the application.

In the past, students have produced many interesting projects like DJ desks, Beat Box systems, drum machines, tutorial applications for guitar students or applications for assisting in music composition. Some of these projects were quite sophisticated and had a high ranking.

4 The Project

It should be noted that this project is more difficult than Assignments 1 and 2. In fact, Assignments 1 and 2 would be considered as “One” projects on the project complexity scale. This is true not just in the scope of the project but a higher standard of software development is also expected.

4.1 Project Management

4.1.1 Time Management

Time management has been the prime problem with Assignment 3 projects in previous semesters. As with projects in industry, people often underestimate the time required to implement functionality. Too often students ask for help too late. Two tips: first, after getting project approval spend some time experimenting with the key aspects of the project (libraries, algorithms, etc) to get a feel for how difficult these are. Second, set some deadlines for particular functionality to be established (think the point of no return). If these are exceeded consider either simplifying the functionality to a fallback position or asking Peter for advice.

4.1.2 Backup and Version Control

As you gradually scale up the development projects you take on - backup and version control of your code becomes more important. At its simplest you may elect to keep progressive copies of your assignment on a USB drive or similar - whenever you reach a coding milestone. There are also more sophisticated backup (such as unison) and version control systems (such as CVS, SVN and git). These typically require a bit more setup. Feel free to talk to us if you are interested.

4.2 Implementation

4.2.1 User Interfaces

We don't expect you to be a user interface expert in this course however keeping the few tips from the checklist in the back of your mind while implementing can help provide a more stylish product.

A lot of user interface design revolves around two things: user expectations and usability. Excluding complete computer beginners, users have interacted with programs under a particular operating system and have implicit expectations of how they behave. Colours, positioning, alignment, type of widgets, keyboard and mouse shortcuts can all have meanings the user associates with particular aspects of programs. When programs have aspects that differ from these expectations, users can find them jarring and either are put off or quit using the program entirely. Typically games have less rigid user expectations than productivity programs, however there may be styles that can be used to align with user expectations. For example, emulating an older style game could lend itself to 8-bit graphics and internal PC speaker beeps. Example projects with excellent game interfaces are located on the course webpage.

Usability is a very large area of which a very tiny portion will be discussed here. Consider Microsoft Word (or any modern word processor) which has been experimenting with usability for many years. There are often several ways to do common actions - Paste can occur from the menu, the toolbar or using the Ctrl-V shortcut. Typically, less experienced users will use the mouse more than power users. Thinking about ways to avoid forcing the user to constantly move from keyboard to mouse again will improve usability.

4.2.2 Code Style

Top-level comment

At the top of every source code file we're looking for a top level comment with your name, the name of the program, current version, what it does and also license/copyright information. In the case of multiple files, there should also be a description of what purpose of the particular file is. Looking at open source code from the internet may give you an idea of what these should look like. Developers that may want to extend your program can look at these comments to work out the state of the project.

Line wrapping

Excluding some unbreakable formulas, each line of code should be no longer than eighty (80) characters. While not apparent on modern widescreen monitors, eighty characters is the width of a standard Linux terminal shell and more

importantly the width of a standard printed page. You should manually break lines in order to make your printed code readable.

Magic Numbers

As a general rule there should be no numbers passed directly to functions. Eg

```
a = 32 * 32
```

In the above example there is no context to indicate the purpose of the number 32. Numbers such as these are referred to as “magic numbers”. These are considered poor style as it makes it more difficult for another programmer to read or indeed for you to read months down the track when you have forgotten the significance of the number. This problem can also occur for any type of variable (there are also magic strings).

Repeated Code

As the amount of code you write grows you will notice that common code fragments will emerge. Periodically look through your code for these fragments and ask yourself “Can I rewrite this so I don’t have multiple bits of code doing the same thing?” Alternately, if you find yourself copying and pasting entire swathes of code stop and think whether you could create a class, procedure or a loop instead. Your ability to generalise code will develop over time as you gain more experience programming.

Spaghetti Code

Somewhat related to the previous section, spaghetti code occurs when the flow of your program is convoluted from an obfuscated class structure or procedures. Again, the more you program, the better your ability to write clear and concise code and avoid the spaghetti.

Plugin Code

To add polish think about how you may be able to create a plugin based system. For example, if designing a game you could have the game levels read in from files so that others can extend your game. How about the characters and story of your game, can that be easily altered? For simulation or analysis tools, how easy is it to plugin a different type of simulation or analysis?

4.3 Documentation

4.3.1 Good Proposals help make good final documents

Even though the proposal is not assessed as such, creating a polished proposal can save time later and serve as a starting point for the final assessable documentation.

Details about what packages you use, how the program works will hopefully require few alterations for your final document. Bedding this down early will allow you to focus on the reflection and installation documentation.

Note that if you included a sketch or mock-up in your initial proposal, we expect you to include a real screen shot for your final design documentation.

4.3.2 Start Early

Often students leave documentation until the last minute in favour of coding last minute features. Don't underestimate the time required for documentation. This assignment requires a detailed project breakdown and the reflective component needs to display a grasp of critical analysis.

4.3.3 Installation

The installation notes you submit are used for us to try out your program. The notes should be about half a page. You should specify:

- the operating system you developed under
- which packages are necessary and whether they require any special installation
- whether your program must be run through IDLE or from the command line and if from the command line whether:
 - your program must be run from a particular directory
 - your program requires particular arguments or flags

I (Peter) am a Linux user and while I don't expect you to develop on my platform of choice I do ask that you take a few simple steps to keep your code relatively platform independent. Please be aware that I run your programs before the demonstrations in order to give maximum feedback possible. Taking these steps will show off your project in the best possible light.

1. Please avoid hardcoded absolute paths. As much as possible keep the directories you use within your root program directory and keep the paths

you specify relative to this root directory. You may also want to have a configuration file separate to your program code where you specify any directories you use so they are easy to change (which you should specify as part of your installation notes).

2. We request that you keep your directory and filenames lowercase and without spaces or special characters. Something like `my_program_name` would be best.
3. Windows considers the files `BaTh.JPG` and `bath.jpg` as the same whereas other operating systems do not. We request that you keep your capitalisation consistent between your files and your code.
4. Instead of writing your paths directly we ask that you use the `os` library to handle these in a platform independent way. See http://www.oreillynet.com/onlamp/blog/2008/01/pymotw_ospath.html
5. Be aware that new lines are either represented as one character or two depending on the operating system. This can be handled seamlessly but using universal newline support: `open('/tmp/hello.txt', 'U').read()`

4.3.4 Reflection

We are interested in what you got out of your project and so we would like you to reflect on what you have achieved. We understand that your project will not be perfect and there will be several things you think could be improved or done in a better way. This document gives you a chance to reflect on all aspects of your project. All known bugs and unwanted features should be clearly listed. You may also discuss issues such as completeness, efficiency, modularity, documentation, usability.

4.4 Demonstration

In the demonstration, I am looking for you to sell your program to me. Show me all the features in the best possible light. In previous semesters it has been possible to secure all possible marks for a particular project band (or occasionally a few more) independent of the quality of other project deliverables.

Typically, you don't need to discuss code with me as part of your demonstration - I will do a code review with you afterwards. You may however choose to talk about code in your demonstration if your project has a very large "behind-the-scenes" component - eg. a physics calculation library, a purely AI program.

Later this semester I will post a message to you asking you to sign up to a demonstration time. The demonstration will be 20 minutes long in which you need to give me or a tutor (not the whole class) a demonstration of your program and we will provide feedback about your program and your code. The

demonstration is also a means for ensuring that you have authored your own work.

During the demonstration we will also be providing constructive criticism about your project and your code. Being apologetic or defensive can detract from your demonstration. We ask that you accept the feedback as it was intended, to help you improve for future projects.

4.4.1 Plan your demo

It's worthwhile to make a brief plan of how to demonstrate your project. Often students are nervous and can become flustered - having a logical and methodical means for showing all your program's functionality can make the demonstration go much smoother. Also think about how you can conclude your demo - "So yeah that's it" probably isn't the best note to end on.

4.4.2 Test under demo conditions

If you decide to use a different computer to that you developed on, be sure to do a "dry run" to make sure the computer is set up properly and the program runs as expected.